
SwigLib
basics

Contents

1 Introduction

The SwigLib project is related to LuaTEX and aims as adding portable library support to this TEX engine without too much fixed binding. The project does not provide Lua code, unless really needed, because it assumes that macro packages have different demands. It also fits in the spirit of TEX and Lua to minimize the core components.

The technical setup is by Luigi Scarso and documentation about how to build the libraries is part of the SwigLib repository. Testing happens with help of the ConTEXt (garden) infrastructure. This short document only deals with usage in ConTEXt but also covers rather plain usage.

todo: reference to Luigi's manual

2 Inside CONTEXT

The recommended way to load a library in ConTEXt is by using the `swiglib` function. This function lives in the global namespace.

```
local gm = swiglib("gmwand.core")
```

After this call you have the functionality available in the `gm` namespace. This way of loading makes ConTEXt aware that such a library has been loading and it will report the loaded libraries as part of the statistics.

If you want, you can use the more ignorant `require` instead but in that case you need to be more explicit.

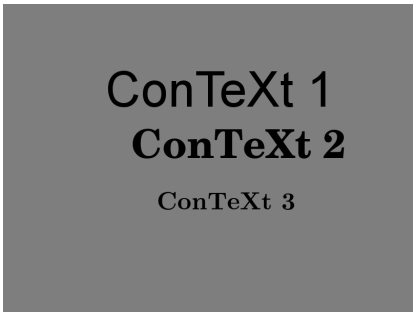
```
local gm = require("swiglib.gmwand.core")
```

Here is an example of using such a library (by Luigi):

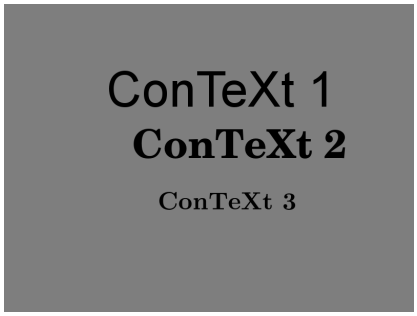
```
\startluacode
local gm       = swiglib("gmwand.core")
local findfile = resolvers.findfile

gm.InitializeMagick(".")

local magick_wand  = gm.NewMagickWand()
```

```
local drawing_wand = gm.NewDrawingWand()
local pixel_wand   = gm.NewPixelWand();

gm.MagickSetSize(magick_wand,800,600)
gm.MagickReadImage(magick_wand,"xc:gray")

gm.DrawPushGraphicContext(drawing_wand)

gm.DrawSetFillColor(drawing_wand,pixel_wand)

gm.DrawSetFont(drawing_wand,findfile("dejavuserifbold.ttf"))
gm.DrawSetFontSize(drawing_wand,96)
gm.DrawAnnotation(drawing_wand,200,200,"ConTeXt 1")

gm.DrawSetFont(drawing_wand,findfile("texgyreschola-bold.otf"))
gm.DrawSetFontSize(drawing_wand,78)
gm.DrawAnnotation(drawing_wand,250,300,"ConTeXt 2")

gm.DrawSetFont(drawing_wand,findfile("lmroman10-bold.otf"))
gm.DrawSetFontSize(drawing_wand,48)
gm.DrawAnnotation(drawing_wand,300,400,"ConTeXt 3")

gm.DrawPopGraphicContext(drawing_wand)

gm.MagickDrawImage(magick_wand,drawing_wand)

gm.MagickWriteImages(magick_wand,"./swiglib-mkiv-gm-1.png",1)
gm.MagickWriteImages(magick_wand,"./swiglib-mkiv-gm-1.jpg",1)
gm.MagickWriteImages(magick_wand,"./swiglib-mkiv-gm-1.pdf",1)

gm.DestroyDrawingWand(drawing_wand)
gm.DestroyPixelWand(pixel_wand)
gm.DestroyMagickWand(magick_wand)
\stopluacode
```

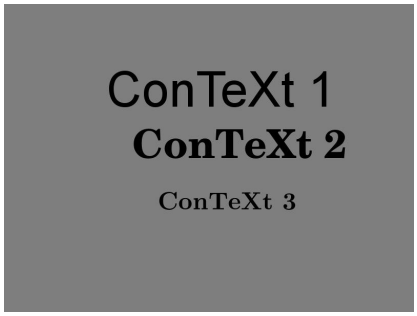
In practice you will probably stay away from manipulating text this way, but it illustrates that you can use the regular ConTEXt helpers to locate files.



png



pdf



jpg

You'd better make sure to use unique filenames for such graphics. Of course a more clever mechanism would only run time consuming tasks once for each iteration of a document.

3 Outside CONTEXT

In the ConTEXt distribution we ship some generic macros and code for usage in plain TEX but there is no reason why they shouldn't work in other macro packages as well. A rather plain example is this:

```
\input luatex-swiglib.tex

\directlua {
    dofile("luatex-swiglib-test.lua")
}

\pdfximage {luatex-swiglib-test.jpg} \pdfrefximage\pdflastximage

\end
```

Assuming that you made the `luatex-plain` format, such a file can be processed using:

```
luatex --fmt=luatex=plain luatex-swiglib-test.tex
```

The loaded Lua file `luatex-swiglib-test.lua` liike like this:

```
local gm = swiglib("gmwand.core")

gm.InitializeMagick(".")

local magick_wand  = gm.NewMagickWand()
local drawing_wand = gm.NewDrawingWand()

gm.MagickSetSize(magick_wand,800,600)
gm.MagickReadImage(magick_wand,"xc:red")
gm.DrawPushGraphicContext(drawing_wand)
gm.DrawSetFillColor(drawing_wand,gm.NewPixelWand())
gm.DrawPopGraphicContext(drawing_wand)
gm.MagickDrawImage(magick_wand,drawing_wand)
gm.MagickWriteImages(magick_wand,"./luatex-swiglib-test.jpg",1)

gm.DestroyDrawingWand(drawing_wand)
gm.DestroyMagickWand(magick_wand)
```

Instead of loading a library with the `swiglib` function, you can also use `require`:

```
local gm = require("swiglib.gmwand.core")
```

Watch the explicit `swiglib` reference. Both methods are equivalent.

4 The libraries

Most libraries are small but some can be rather large and have additional files. This is why we keep them separated. On my system they are collected in the platform binary tree:

```
e:/tex-context/tex/texmf-mswin/bin/lib/luatex/lua/swiglib/gmwand
e:/tex-context/tex/texmf-mswin/bin/lib/luatex/lua/swiglib/mysql
e:/tex-context/tex/texmf-mswin/bin/lib/luatex/lua/swiglib/....
```

One can modulate on this:

```
...tex/texmf-mswin/bin/lib/luatex/lua/swiglib/mysql/core.dll
...tex/texmf-mswin/bin/lib/luajittex/lua/swiglib/mysql/core.dll
...tex/texmf-mswin/bin/lib/luatex/context/lua/swiglib/mysql/core.dll
```

are all valid. When versions are used you can provide an additional argument to the `swiglib` loader:

```
tex/texmf-mswin/bin/lib/luatex/lua/swiglib/mysql/5.6/core.dll
```

This works with:

```
local mysql = swiglib("mysql.core","5.6")
```

as well as:

```
local mysql = swiglib("mysql.core")
```

It is hard to predict how operating systems look up libraries and especially nested loads, but as long as the root of the `swiglib` path is known to the file search routine. We've kept the main conditions for success simple: the core library is called `core.dll` or `core.so`. Each library has an (automatically called) initialize function named `luaopen_core`. There is no reason why (sym)links from the `swiglib` path to someplace else shouldn't work.

In `texmfcnf.lua` you will find an entry like:

```
CLUAINPUTS = ".;$SELFAUTOLOC/lib/{$engine/context,$engine}/lua//"
```

Which in practice boils down to a search for `luatex` or `luajittex` specific libraries. When both binaries are compatible and there are no `luajittex` binaries, the regular `luatex` libraries will be used.

The `swiglib` loader function mentioned in previous sections load libraries in a special way: it changes dir to the specific path and then loads the library in the usual way. After that it returns to the path where it started out. After this, when the library needs additional libraries (and for instance graphicmagick needs a lot of them) it will first look on its own path (which is remembered).

The MkIV lookups are somewhat more robust in the sense that they first check for matches on engine specific paths. This comes in handy when the search patterns are too generic and one can match on for instance `luajittex` whilc `luatex` is used.

5 Colofon

author Hans Hagen, PRAGMA ADE, Hasselt NL
version May 15, 2015
website www.pragma-ade.nl – www.contextgarden.net

comment the swiglib infrastructure is implemented by Luigi Scarso